Bayesian Decoding of Substitution Cipher

Paul Zhou

Department of Applied Mathematics, Brown University, Providence, RI, 02192

Abstract

This paper discusses a Bayesian decoding method of substitution cipher. Maximum Likelihood method is used to match the frequency of symbols in the encoded text to that of natural English, and the maximization problem is solved by a Markov Chain Monte Carlo algorithm.

Keywords: Substitution Cipher Decoding, Markov Chain Monte Carlo, Maximum Likelihood, Metropolis Algorithm

1. Introduction

Substitution Cipher is a text encryption method that encodes plain text by swapping each symbol by another symbol, transforming the original *plaintext* into *ciphertext*. It is a relatively simple method of encrypting intelligible text into unintelligible text, and has been in use for hundreds of years [1]. Some of these encoded messages carry important information, so it is important to have a reliable Mathematical method of solving these ciphers.

An interesting and reliable approach was first proposed by Marc Coram and Phil Beineke in the Stanford statistical consulting service [2], and later investigated with more detail by Stephen Conner [3]. Coram proposed breaking the substitution cipher with frequency analysis – figuring out the frequencies each individual letter and each pairs of letters occur in natural English language. He then used the frequencies as a basis for a Markov Chain Monte Carlo (MCMC) [4] algorithm that successfully found the substitution code that corresponds best to these frequencies.

In this paper, I will follow the foot steps of Coram and recreate his decoding algorithm. I will model the natural English language as a Markov Chain and perform Bayesian inference on the substitution code with the maximum likelihood method, and solve the maximization problem using MCMC. The

rest of the paper is organized as follows: First (Section 2), I will formulate the substitution cipher problem in precise Mathematical language and introduce related background knowledge for solving it. Then, in Section 3, I will lay out the decoding algorithm in detailed steps. Finally, I will use the algorithm to break three substitution ciphers in practice and discuss the results (Sections 4 and 5)

2. Problem Setup and Background

In this section, I will formulate the decoding problem and introduce background knowledge on Bayesian inference and MCMC.

2.1. Scrambling and Decoding

Given a piece of text, we can encrypt it with substitution cipher by scrambling it. I consider the English language to be a sequence of 27 distinct symbols: 26 characters $a - z$ and _ (space character). Let $S = \{a, b, c, ..., z, \_\}$ be the set of these 27 unit characters. A substitution cipher is a one-to-one function

$$\sigma : S \to S, \quad \sigma \in S_{27}$$

where S_{27} denotes the space of all permutations for a set of size 27. A substitution cipher σ is essentially a permutation of the 27 legal characters, where each character in the *plaintext* (the original intelligible text) has a one-to-one mapping with a character in the *ciphertext* (the encoded unintelligible text). As a result, I will use the expressions "permutation" and "substitution cipher" interchangeably. Given plaintext sequence $a_1a_2...a_n$, substitution cipher σ will encoded it into an equal-length sequence $b_1b_2...b_n$ where $b_i = \sigma(a_i)$.

The decoding problem is: given a sequence of ciphertext $b_1b_2...b_n$, how do we find the correct cipher key (permutation) σ_* to map it back to plaintext $a_1a_2...a_n = \sigma_*^{-1}(b_1)\sigma_*^{-1}(b_2)...\sigma_*^{-1}(b_n)$.

2.2. Digram Model of the English Language

In order to formalize the decoding problem, we have to model the English language precisely. For decoding purposes, I take the simplifying assumption that language is just a sequence of discrete symbols. Following Shannon's work [5], I *assume* that language is a stationary stochastic process with state space $S = \{a, b, c, ..., z, \_\}$ and it can be approximated by a Markov

Chain. I will refer to language that satisfies these two assumptions as *true language*. For simplicity, I assume a *digram* model of English: the occurrence of each symbol is determined only by the previous symbol according to some probability distribution. This comes from the Markov assumption of true language. More precisely,

$$P(a_t|a_1a_2...a_{t-1}) = P(a_t|a_{t-1}) \quad t = 1, 2, 3, ...$$

In other worlds, we consider the English language to be consisting of tuples of symbols, and the latter symbol is determined by the former according to some transition distribution. These transition probabilities are encoded by a square transition matrix Q, where $Q(x, y)$ is the probability symbol x will transition into symbol y in a language sequence. Therefore, each row of the transition matrix Q is a probability distribution:

$$\begin{cases} \sum_y Q(x, y) = 1, & \forall x \in S \\ Q(x, y) \geq 0, & \forall x, y \in S \end{cases}$$

The starting character in a language sequence, a_1, is drawn according to some one-point probability distribution $P_{one} : S \to \mathbb{R}$. This is the one-point distribution that encodes how often each symbol occurs in the natural English language.

Following this digram model, we can model the English language as:

$$\begin{aligned} P(a_1a_2a_3...a_n) &= P(a_1) \cdot P(a_2|a_1) \cdot P(a_3|a_1a_2) \cdot P(a_4|a_1a_2a_3) \\ &\qquad \cdots P(a_n|a_1a_2...a_{n-1}) \\ &= P(a_1) \cdot P(a_2|a_1) \cdot P(a_3|a_2) \cdot P(a_4|a_3) \cdots P(a_n|a_{n-1}) \\ &= P_{one}(a_1) \cdot Q(a_1, a_2) \cdot Q(a_2, a_3) \cdots Q(a_{n-1}, a_n) \end{aligned} \tag{1}$$

2.3. Bayesian Inference

Given the ciphertext $b_1b_2...b_n$, I find the most likely substitution cipher by performing Bayesian inference on σ. There are typically three steps in a Bayesian inference process:

1. making a probabilistic model of the *prior*: in the case of decoding, I have to model the true language $a_1a_2...a_n$ as well as the permutation σ. The prior is $P(a_1a_2...a_n, \sigma)$
2. express the *posterior* $P(\sigma|a_1a_2...a_n)$ using Bayes Rule.
3. maximum likelihood approach: find the σ that maximizes the posterior.

A more detailed approach of these steps is presented in Section 3.

2.4. MCMC and the Metropolis Algorithm

Finding the σ_* that maximizes the posterior is a hard maximization problem, because σ belongs to a huge state space S_{27}. Naively iterating over all σ to find the maximum is computationally intractable. Therefore, I resort to solving the problem probabilistically using Markov Chain Monte Carlo (MCMC) methods.

MCMC is a family of algorithms for sampling from a probability distribution. It performs the sampling by constructing a Markov Chain that has the desired probability distribution as the equilibrium distribution, and so samples of that distribution can be obtained by simply recording states from the Markov Chain. Here, I use MCMC as a way of sampling the distribution of the permutation σ.

Specifically, I use the Metropolis algorithm [6] for sampling. The algorithm assumes a state space S_m, an energy function $E : S_m \to \mathbb{R}$, and an inverse temperature hyperparameter β. The state space S_m is constructed into a graph G and the algorithm performs a biased walk on G:

Algorithm 1: Metropolis Algorithm

Input: E, β, S_m, number of total steps T, start node $x_1 \in G$

Output: Markov chain $x_1, x_2, ..., x_T$

1 record current node as a state in the resulting Markov Chain

2 steps taken += 1

3 use an unbiased random walk to choose a neighbor y

4 accept/reject rule:

(I) If $\Delta E = E(y) - E(x) < 0$, then move current node from x to y.

(II) If $\Delta E \geq 0$, accept the move from x to y with probability $e^{-\beta \Delta E}$, or else stay at x.

Repeat *Steps 1 to 4*, until steps taken $> T$

This algorithm will output a Markov chain $\{X_t\} = \{x_1, x_2, ..., x_T\}$ which will eventually converge to the true hidden permutation σ_*. The reasons for this convergence will be explained in Section 3. In the end, σ_* is the answer to the decoding problem.

3. Computational Methods

First, I will use the Bayesian inference process to find the likelihood function of permutation σ; then, I will solve the maximum likelihood problem using the Metropolis algorithm.

In order the compute the prior, we must model true language $a_1a_2...a_n$ and the distribution of the permutations σ.

Following Equation 1, I model true language using a digram model:

$$P_{true}(a_1a_2...a_n) = P_{one}(a_1) \cdot Q(a_1, a_2) \cdot Q(a_2, a_3) \cdots Q(a_{n-1}a_n)$$

where P_{one} is the one-point distribution and Q is the transition matrix (Section 2). Although we don't have access to the true P_{one} and Q, we can approximate it with data. Given a text corpus of natural English $\{p_i\}_{i=1,2,...,N}$ consisting of symbols in $S = \{a, b, c, ...z, \_\}$, we can compute

$$\begin{cases} P_{one}(x) = \frac{\sum_{i=1}^{N} \mathbb{1}(p_i=x)}{N} \\ Q(x,y) = \frac{\sum_{i=1}^{N-1} \mathbb{1}(p_i=x, p_{i+1}=y)}{\sum_{i=1}^{N-1} \mathbb{1}(p_i=x)} \end{cases} \tag{2}$$

for all $x, y \in S$, where $\mathbb{1}(\cdot)$ is the identity function.

Next, I will model the distribution of the permutation σ as a uniform distribution on S_{27}:

$$P(\sigma = \sigma_i) = \frac{1}{27!}, \quad \forall \sigma_i \in S_{27}.$$

This is the natural design choice given we don't have any information on the apriori distribution of σ. Moreover, this distribution is *independent* of the digram language model.

As a result, the prior will be:

$$\begin{aligned} P(a_1a_2...a_n, \sigma) &= P_{true}(a_1a_2...a_n) \cdot P(\sigma) \\ &= P_{true}(a_1a_2...a_n) \cdot \frac{1}{27!} \\ &= P_{one}(a_1)Q(a_1, a_2)Q(a_2, a_3) \cdots Q(a_{n-1}, a_n) \cdot \frac{1}{27!} \end{aligned} \tag{3}$$

The posterior is:

$$\begin{aligned}
P(\sigma|b_1b_2...b_n) &= \frac{P(b_1b_2...b_n,\sigma)}{P(b_1b_2...b_n)} \\
&= \frac{P(b_1b_2...b_n|\sigma)\cdot P(\sigma)}{P(b_1b_2...b_n)} \\
&= \frac{P_{true}(\sigma^{-1}(b_1)\sigma^{-1}(b_2)...\sigma^{-1}(b_n))\cdot \frac{1}{27!}}{P(b_1b_2...b_n)} \\
&= P_{true}(\sigma^{-1}(b_1)\sigma^{-1}(b_2)...\sigma^{-1}(b_n))\cdot \frac{1/27!}{P(b_1b_2...b_n)} \\
&= P_{one}(\sigma^{-1}(b_1))\cdot Q(\sigma^{-1}(b_1),\sigma^{-1}(b_2))\cdot\cdot \\
&\qquad Q(\sigma^{-1}(b_{n-1}),\sigma^{-1}(b_n))\cdot \frac{1/27!}{P(b_1b_2...b_n)} \\
&\propto P_{one}(\sigma^{-1}(b_1))\cdot Q(\sigma^{-1}(b_1),\sigma^{-1}(b_2))\cdot\cdot \\
&\qquad Q(\sigma^{-1}(b_{n-1}),\sigma^{-1}(b_n))
\end{aligned} \tag{4}$$

This gives the likelihood function

$$L(\sigma) = P_{one}(\sigma^{-1}(b_1))\cdot Q(\sigma^{-1}(b_1),\sigma^{-1}(b_2))\cdots Q(\sigma^{-1}(b_{n-1}),\sigma^{-1}(b_n)) \tag{5}$$

The Maximum Likelihood approach suggests that the correct answer is the one that maximizes the posterior/likelihood:

$$\begin{aligned}
\sigma_* &= \arg\max_\sigma P(\sigma|b_1b_2...b_n) \\
&= \arg\max_\sigma L(\sigma) \\
&= \arg\max_\sigma \log L(\sigma) \\
&= \arg\min_\sigma -\log L(\sigma) \\
&= \arg\min_\sigma -\log(P_{one}(\sigma^{-1}(b_1))\cdot Q(\sigma^{-1}(b_1),\sigma^{-1}(b_2))\cdot\cdot \\
&\qquad Q(\sigma^{-1}(b_{n-1}),\sigma^{-1}(b_n))) \\
&= \arg\min_\sigma -\log P_{one}(\sigma^{-1}(b_1)) - \log Q(\sigma^{-1}(b_1),\sigma^{-1}(b_2)) - \\
&\qquad ... - \log Q(\sigma^{-1}(b_{n-1}),\sigma^{-1}(b_n))
\end{aligned} \tag{6}$$

I will do solve this minimization problem with the Metropolis algorithm. First, define a graph $G = (V, E)$ on the space of permutations S_{27}. Each

node will be one permutation ($V = S_{27}$) and two nodes σ_i, σ_j are connected by an edge if σ_i, σ_j differ only by the swapping of one pair of symbols (e.g. $abcd...z\_ \to \_z...dcba$ and $abcd...z\_ \to \_z...dcab$ differ only by swapping a and b).

Further define energy function for the permutations

$$E(\sigma) = -\log L(\sigma). \tag{7}$$

Minimizing this energy will equivalently solve the maximum likelihood problem (Equation 6). The Metropolis algorithm (algorithm 1) provides a way of doing a walk over the graph such that eventually will converge to staying at low energy nodes (notice how step 4 encourages movement towards low energy nodes). The convergence lowest-energy node σ_* is the permutation that maximizes the likelihood, and therefore represents the hidden substitution cipher.

Using the cipher code σ_*, we can easily decode the ciphertext $b_1 b_2 ... b_n$:

$$a_1 a_2 ... a_n = \sigma_*^{-1}(b_1)\sigma_*^{-1}(b_2)...\sigma_*^{-1}(b_n)$$

4. Experiments

I test the decoding method described above using three pieces of encoded texts, all of which are included in Appendix B. All experiments are carried out in Python, and and code is attached in Appendix C.

To compute the prior distribution of the true language, I used the book of Pride and Prejudice by Jane Austin [7] as a text corpus for natural English language. The text is converted to all lowercase letters and filtered to only contain symbols in our defined state space $S = \{a, b, ..., z, \_\}$. Then Equation 2 is used to data-mine the one-point distribution and transition matrix of the symbols. When encountering a transition $x \to y$ with $Q(x, y) = 0$, I set $Q(x, y) = 10^{-16}$ to prevent numerical issues when taking the logarithm in Equation 7.

For implementing the Metropolis algorithm, the energy is defined as in Equations 7 and 5. The state space is the space of all permutations S_{27}, and the inverse temperature parameter is taken to be $\beta = 1$. The algorithm starts at an initial "identity" permutation $\sigma_1 : abc...z\_ \to abc...z\_$ and takes a total of $T = 100,000$ steps.

5. Results and Discussion

5.1. Decoding Results

Using my implementation of the decoding method discussed above, I have obtained the decoded messages for all three encryption. Excerpts of the decoded messages are presented below, and the full-length decoded messages are included in Appendix A. The results show that this Bayesian method with MCMC can break substitution ciphers with minimal computing power and time.

For code f_45.txt, the plaintext is an excerpt from Chapter 5 of Feynman Lectures On Computation [8]:

> i would now like to take a look at a subject which is extremely interesting but almost entirely academic in nature this is the subject of the energetics of computing we want to address the question how much energy must be used in carrying out a computation
>
> ...
>
> this is actually what we mean by isothermal compression we do the compression slowly ensuring that at all times the gas and the surrounding bath are in thermal equilibrium

For code h_45.txt, the plaintext is an excerpt from Chapter 1 of Harry Potter and the Sorcerer's Stone:

> mr and mrs dursley of number four privet drive were proud to say that they were perfectly normal thank you very much they were the last people youd expect to be involved in anything strange or mysterious because they just didnt hold with such nonsense
>
> ...
>
> mr dursley however had a perfectly normal owl free morning he yelled at five different people he made several important telephone

For code j_45.txt, the plaintext is an excerpt from Chapter 1 of Finnegans Wake by James Joyce:

> riverrun past eve and adams from swerve of shore to bend of bay brings us by a commodius vicus of recirculation back to howth castle and environs sir tristram violer damores frover the short sea had passencore

rearrived from north armorica on this side the scraggy isthmus of europe minor to wielderfight his penisolate war

...

mister finn youre going to be mister finnagain comeday morm and o youre vine senddays eve and ah youre vinegar hahahaha mister funn youre going to be fined again

The three substitution cipher keys are presented in Table 1.

5.2. Visualization of the Digram Language Model

The digram model of English is completely determined by P_{one} and Q, both of which are obtained by data mining and visualized in Figure 1. The figures show that letter 'e' and the space symbol occur very frequently in natural English, and the other symbols often transition into these two characters. This corresponds with one's common sense about English: 'e' is a common vowel and space occurs very frequently because English words are almost all shorter than 27 letters. Other popular symbols include 'a', 'n', and 'i'. This approximate distribution of the symbols is also corroborated by Li and Miramontes [9] and New and Grainger [10].

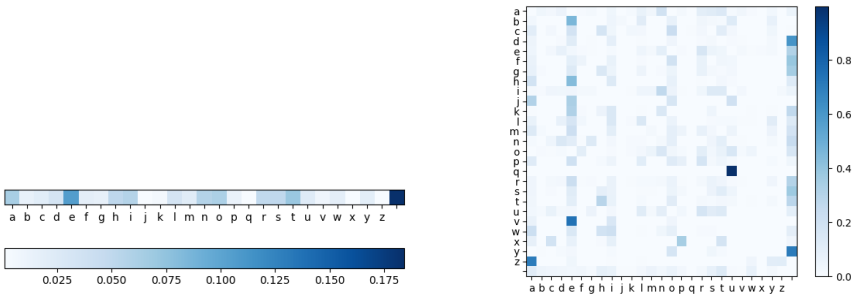

Figure 1: Left: visualization of the one-point distribution P_{one}. Right: visualization of the transition matrix Q of the 27 symbols

Even though the digram model is very simple and is almost definitely not powerful enough to model real English, experiments have proven that it is sufficient for decoding substitution ciphers.

permutation / cipher key			
plaintext	f code	h code	j code
a	b	_	x
b	w	v	c
c	r	p	b
d	o	k	l
e	g	c	s
f	y	y	v
g	i	o	i
h	e	r	j
i	m	h	q
j	p	i	m
k	u	q	w
l	q	m	e
m	j	d	t
n	x	x	n
o	v	j	g
p	f	n	p
q	h	u	h
r	d	f	f
s	n	l	z
t	t	t	o
u	c	s	_
v	z	b	k
w	s	g	r
x	_	a	a
y	a	z	d
z	k	e	y
_	l	w	u

Table 1: The key for the three substitution ciphers. For example, plaintext letter 'a' is encoded as 'b' in the f code, '_' (space) in the h code, and 'x' in the j code.

5.3. Energy change in Metropolis

As the Metropolis algorithm runs, the walk on the graph G should converge to nodes with the lowest energy. Figure 2 confirm that energy of nodes lowers as the number of step taken increases.

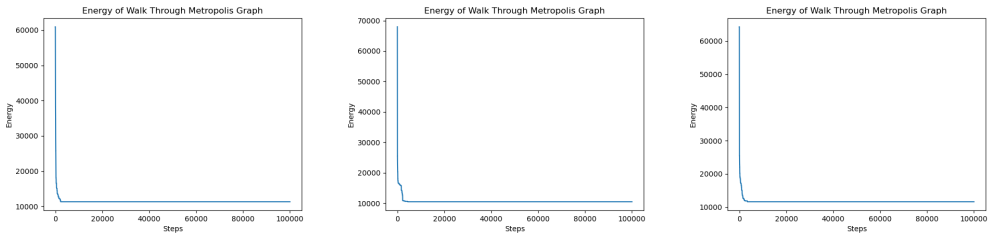

Figure 2: The energy of the nodes encountered during the walk over the Metropolis graph. From left to right, the walks are over the f code, h code, and j code, respectively.

The energy very quickly converges to the minimum as the walk on the Metropolis graph goes on. In fact, minimum energy is reached at around $5,000$ steps for all three codes. This suggests that the hidden permutation can be found by running the Metropolis algorithm for only $5,000$ steps, instead of $100,000$. To confirm this hypothesis, I ran an experiment only executing Metropolis for $5,000$ steps and successfully decoded all three codes.

Therefore, we could propose an improvement over Algorithm 1. Instead of running the Metropolis algorithm for a fixed number of steps T, we can terminate the algorithm by checking for convergence of energy. We can define convergence to be when the energy of the current node does not change for $3,000$ consecutive steps.

5.4. Complexity of Decoding

There are many factors that might influence the ease of which a scrambled text can be decoded.

One of characteristics that made decoding the three texts possible is that the encoded texts are sufficiently long. The three pieces of texts have 4570, 4426, and 4280 symbols, respectively. If the encoded texts had significantly fewer symbols, decoding with this method might not be possible.

Take an extreme case as a thought experiment: the ciphertext only has one symbol p. In this case, the most informed guess we can make is that p encodes the most commonly occurring symbol in natural English – the space character. As for what the other 26 symbols encode, we can only make

random guesses without any information. Even though we can arrive at a maximum likelihood answer easily, there is a very small chance this answer is the correct one.

More mathematically, fewer symbols in the ciphertext means the likelihood function (Equation 5) $L(\sigma)$ would have fewer terms, and so the energy function $E(\sigma) = -\log L(\sigma)$ would be scales smaller. This makes it harder to identify the σ that minimizes the energy, because less data means the energy is less accurate. With a noisy energy estimation, we may converge to a wrong permutation that has the least empirical energy but not the least real energy.

Another factor that influences the decoding complexity is the semantic similarity between the target encoded text and the source text used to obtain P_{one} and Q. The more similar the source and target tasks are, the easier it is to decode.

As another thought experiment, take the source text to be a Shakespeare play, and the target text to be "text speak" taken from online sources on the internet in 2022. While the source text is filled with Elizabethan English, the target text is modern English sprinkled with newly invented acronyms. The transitional probabilities P_{one} and Q must be different for these two types of texts, and so trying to match them assuming they're the same must not work very well.

In my experiments, the source text is a 19^{th} century novel, while the three encoded texts are a $1960s$ Physics textbook, a $90s$ fantasy novel, and a 1939 novel by an Irish writer, respectively. While the first two encoded texts encode standard English, the last one is written largely in idiosyncratic language, which blends standard English words with made-up words in multiple languages. So while the source text is similar to the first two encoded texts, it is different from the third one to some degree. Therefore, it can be expected that the third text (j code) is harder to decode than its two counterparts.

To test the difficulty of decoding different encoded texts, I propose running the algorithm with different random seeds and recording the success rates. A lower success rate would indicate a harder decoding problem. Through changing the random seeds, I change how the random walk is ordered in the Metropolis algorithm by controlling how the neighboring node is chosen. For all 10 random seeds I used, f code, h code, and j code has decoding success rates 90%, 80%, and 50%. This suggests that j code is much harder to decode, because it is more brittle to the order of the nodes encountered in the random walk. It is also interesting to note that none of

the nodes had a success rate of 100%, suggesting that the algorithm in this paper is not always reliable.

5.5. Run Time of the Program

On a Linux machine using 1 CPU core, taking $100,000$ steps using the Metropolis algorithm takes about 15 minutes, and taking $5,000$ steps takes less than 1 minute. Therefore, it only takes less than 1 minute each to decode the three ciphers, which is very efficient.

5.6. Choices of Parameters

The parameters that can vary in the algorithm are: the random seed, the inverse temperature β, number of total steps T, and the start node $x_1 \in G$. The change of random seed and T are already addressed above, and I will address the effects of β and x_1 here.

The parameter β controls the probability of moving to a higher-energy node: $e^{-\beta E}$ with $E \geq 0$. A larger β value corresponds to a smaller probability of jumping to a higher-energy node. Of

$$\beta \in \{0.001, 0.01, 0.1, 0.5, 1, 5, 10, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7\}$$

, $\beta = \{0.001, 0.01\}$ were not able to decode the f text. It is expected that small β won't be able to break the substitution cipher: when the probability of moving is too big, the random walk won't stay at the minimum and never converges, or it might never find the minimum because of a tendency to move towards high-energy nodes. However, it is a bit surprising that big β is still able to successfully decode. When β is too large, there is no chance of jumping to a higher-energy node, and so the random walk might get stuck in a local minimum and never finds the global minimum. The reason that decoding is still successful is that the distribution of σ in a unimodal distribution, and doesn't suffer from being stuck at local extremums.

The parameter x_1 controls the starting position of the random walk on graph G. I ran 10 experiments with different starting positions, and all of them converged quickly and successfully decoded the f text. It seems that x_1 does not have a big effect on the decoding algorithm, so it can be chosen at random.

6. Conclusion

In this paper, I have presented a method for decoding substitution ciphers with Maximum Likelihood method and the Metropolis algorithm. Evaluation using three empirical ciphers show that the algorithm is effective in most cases.

References

[1] S. Singh, The code book, volume 7, Doubleday New York, 1999.

[2] P. Diaconis, The markov chain monte carlo revolution, Bulletin of the American Mathematical Society 46 (2009) 179–205.

[3] S. Conner, Simulation and solving substitution codes, Master's thesis, Department of Statistics, University of Warwick (2003).

[4] A. E. Gelfand, A. F. Smith, Sampling-based approaches to calculating marginal densities, Journal of the American statistical association 85 (1990) 398–409.

[5] C. E. Shannon, A mathematical theory of communication, ACM SIGMOBILE mobile computing and communications review 5 (2001) 3–55.

[6] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, E. Teller, Equation of state calculations by fast computing machines, The journal of chemical physics 21 (1953) 1087–1092.

[7] J. Austen, Pride and prejudice, Broadview Press, 2001.

[8] R. P. Feynman, T. Hey, R. W. Allen, Feynman lectures on computation, CRC Press, 2018.

[9] W. Li, P. Miramontes, Fitting ranked english and spanish letter frequency distribution in us and mexican presidential speeches, Journal of Quantitative Linguistics 18 (2011) 359–380.

[10] B. New, J. Grainger, On letter frequency effects, Acta Psychologica 138 (2011) 322–328.

Appendix A. Full Decoded Messages

Appendix A.1. Decoded f_45.txt

i would now like to take a look at a subject which is extremely interesting but almost entirely academic in nature this is the subject of the energetics of computing we want to address the question how much energy must be used in carrying out a computation this doesnt sound all that academic after all a feature of most modem machines is that their energy consumption when they run very fast is quite considerable and one of the limitations of the fastest machines is the speed at which we can drain off the heat generated in their components such as transistors during operation the reason i have described our subject as academic is because we are actually going to ask another of our fundamental questions what is the minimum energy required to carry out a computation to introduce these more physical aspects of our subject i will return to the field covered in the last chapter namely the theory of information it is possible to treat this subject from a strictly physical viewpoint and it is this that will make the link with the energy of computation to begin with i would like to try to give you an understanding of the physical definition of the information content of a message that physics should get involved in this area is hardly surprising remember shannon was initially interested in sending messages down real wires and we cannot send messages of any kind without some interference from the physical world i am going to illustrate things by concentrating on a particular very basic physical model of a message being sent i want you to visualize the message coming in as a sequence of boxes each of which contains a single atom in each box the atom can be in one of two places on the left or the right side if its on the left that counts as a zero bit if its on the right its a one so the stream of boxes comes past me and by looking to see where each atom is i can work out the corresponding bit to see how this model can help us understand information we have to look at the physics of jiggling atoms around this requires us to consider the physics of gases so i will begin by taking a few things i need from that let us begin by supposing we have a gas containing n atoms or molecules occupying a volume v one we will take this gas to be an exceptionally simple one each atom or molecule within it we take the terms to be interchangeable here is essentially free there are no forces of attraction or repulsion between each constituent this is actually a good approximation at moderately low pressures i am now going to shrink the gas pushing against its volume with a piston compressing it to volume v two i do all this isothermally that is i

immerse the whole system in a thermal bath at a fixed temperature t so that the temperature of my apparatus remains constant isnt it wonderful that this has anything to do with what were talking about im going to show you how first we want to know how much work w it takes to compress the gas now a standard result in mechanics has it that if a force f moves through a small distance dx the work done dw is fdx if the pressure of the gas is p and the cross sectional area of the piston is a we can rewrite this using f equals pa and letting the volume change of the gas dv equals adx so that dw is pdv now we draw on a standard result from gas theory for an ideal gas at pressure p volume v and temperature t we have the relation pv equals nkt where n is the number of molecules in the gas and k is boltzmanns constant as t is constant our isothermal assumption we can perform a simple integration to find w since v two is smaller than v one this quantity is negative and this is just a result of the convention that work done on a gas rather than by it has a minus sign now ordinarily when we compress a gas we heat it up this is a result of its constituent atoms speeding up and gaining kinetic energy however in our case if we examine the molecules of the gas before and after compression we find no difference there are the same number and they are jiggling about no more or less energetically than they were before there is no difference between the two at the molecular level so where did the work go we put some in to compress the gas and conservation of energy says it had to go somewhere in fact it was converted into internal gas heat but was promptly drained off into the thermal bath keeping the gas at the same temperature this is actually what we mean by isothermal compression we do the compression slowly ensuring that at all times the gas and the surrounding bath are in thermal equilibrium

Appendix A.2. Decoded h_45.txt

mr and mrs dursley of number four privet drive were proud to say that they were perfectly normal thank you very much they were the last people youd expect to be involved in anything strange or mysterious because they just didnt hold with such nonsense mr dursley was the director of a firm called grunnings which made drills he was a big beefy man with hardly any neck although he did have a very large mustache mrs dursley was thin and blonde and had nearly twice the usual amount of neck which came in very useful as she spent so much of her time craning over garden fences spying on the neighbors the dursleys had a small son called dudley and in their opinion there was no finer boy anywhere the dursleys had everything they wanted

but they also had a secret and their greatest fear was that somebody would discover it they didnt think they could bear it if anyone found out about the potters mrs potter was mrs dursleys sister but they hadnt met for several years in fact mrs dursley pretended she didnt have a sister because her sister and her good for nothing husband were as undursleyish as it was possible to be the dursleys shuddered to think what the neighbors would say if the potters arrived in the street the dursleys knew that the potters had a small son too but they had never even seen him this boy was another good reason for keeping the potters away they didnt want dudley mixing with a child like that when mr and mrs dursley woke up on the dull gray tuesday our story starts there was nothing about the cloudy sky outside to suggest that strange and mysterious things would soon be happening all over the country mr dursley hummed as he picked out his most boring tie for work and mrs dursley gossiped away happily as she wrestled a screaming dudley into his high chair none of them noticed a large tawny owl flutter past the window at half past eight mr dursley picked up his briefcase pecked mrs dursley on the cheek and tried to kiss dudley good bye but missed because dudley was now having a tantrum and throwing his cereal at the walls little tyke chortled mr dursley as he left the house he got into his car and backed out of number fours drive it was on the corner of the street that he noticed the first sign of something peculiar a cat reading a map for a second mr dursley didnt realize what he had seen then he jerked his head around to look again there was a tabby cat standing on the corner of privet drive but there wasnt a map in sight what could he have been thinking of it must have been a trick of the light mr dursley blinked and stared at the cat it stared back as mr dursley drove around the corner and up the road he watched the cat in his mirror it was now reading the sign that said privet drive no looking at the sign cats couldnt read maps or signs mr dursley gave himself a little shake and put the cat out of his mind as he drove toward town he thought of nothing except a large order of drills he was hoping to get that day but on the edge of town drills were driven out of his mind by something else as he sat in the usual morning traffic jam he couldnt help noticing that there seemed to be a lot of strangely dressed people about people in cloaks mr dursley couldnt bear people who dressed in funny clothes the getups you saw on young people he supposed this was some stupid new fashion he drummed his fingers on the steering wheel and his eyes fell on a huddle of these weirdos standing quite close by they were whispering excitedly together mr dursley was enraged to see that a couple of them werent young at all why that man had to be older

than he was and wearing an emerald green cloak the nerve of him but then it struck mr dursley that this was probably some silly stunt these people were obviously collecting for something yes that would be it the traffic moved on and a few minutes later mr dursley arrived in the grunnings parking lot his mind back on drills mr dursley always sat with his back to the window in his office on the ninth floor if he hadnt he might have found it harder to concentrate on drills that morning he didnt see the owls swooping past in broad daylight though people down in the street did they pointed and gazed open mouthed as owl after owl sped overhead most of them had never seen an owl even at nighttime mr dursley however had a perfectly normal owl free morning he yelled at five different people he made several important telephone

Appendix A.3. Decoded j_45.txt

riverrun past eve and adams from swerve of shore to bend of bay brings us by a commodius vicus of recirculation back to howth castle and environs sir tristram violer damores frover the short sea had passencore rearrived from north armorica on this side the scraggy isthmus of europe minor to wielderfight his penisolate war nor had topsawyers rocks by the stream oconee exaggerated themselse to laurens countys gorgios while they went doublin their mumper all the time nor avoice from afire bellowsed mishe mishe to tauftauf thuartpeatrick not yet though venissoon after had a kidscad buttended a bland old isaac not yet though alls fair in vanessy were sosie sesthers wroth with twone nathandjoe rot a peck of pas malt had jhem or shen brewed by arclight and rory end to the regginbrow was to be seen ringsome on the aquaface the fall bababadalgharaghtakamminarronnkonnbronntonner ronntuonnthunntrovarrhounawnskawntoohoohoordenenthurnuk of a once wallstrait oldparr is retaled early in bed and later on life down through all christian minstrelsy the great fall of the offwall entailed at such short notice the pftjschute of finnegan erse solid man that the humptyhillhead of humself prumptly sends an unquiring one well to the west in quest of his tumptytumtoes and their upturnpikepointandplace is at the knock out in the park where oranges have been laid to rust upon the green since devlinsfirst loved livvy what clashes here of wills gen wonts oystrygods gaggin fishy gods brekkek kekkek kekkek kekkek koax koax koax ualu ualu ualu quaouauh where the baddelaries partisans are still out to mathmaster malachus micgranes and the verdons catapelting the camibalistics out of the whoyteboyce of hoodie head assiegates and boomeringstroms sod s brood be me fear sanglorians save arms apeal

with larms appalling killykillkilly a toll a toll what chance cuddleys what cashels aired and ventilated what bidimetoloves sinduced by what tegotetabsolvers what true feeling for their s hayair with what strawng voice of false jiccup o here here how hoth sprowled met the duskt the father of fornicationists but o my shining stars and body how hath fanespanned most high heaven the skysign of soft advertisement but was iz iseut ere were sewers the oaks of ald now they lie in peat yet elms leap where askes lay phall if you but will rise you must and none so soon either shall the pharce for the nunce come to a setdown secular phoenish bygmester finnegan of the stuttering hand freemen s maurer lived in the broadest way immarginable in his rushlit toofarback for messuages before joshuan judges had given us numbers or helviticus committed deuteronomy one yeastyday he sternely struxk his tete in a tub for to watsch the future of his fates but ere he swiftly stook it out again by the might of moses the very water was eviparated and all the guenneses had met their exodus so that ought to show you what a pentschanjeuchy chap he was and during mighty odd years this man of hod cement and edifices in toper s thorp piled buildung supra buildung pon the banks for the livers by the soangso he addle liddle phifie annie ugged the little craythur wither hayre in honds tuck up your part inher oftwhile balbulous mithre ahead with goodly trowel in grasp and ivoroiled overalls which he habitacularly fondseed like haroun childeric eggeberth he would caligulate by multiplicables the alltitude and malltitude until he seesaw by neatlight of the liquor wheretwin twas born his roundhead staple of other days to rise in undress maisonry upstanded joygrantit a waalworth of a skyerscape of most eyeful hoyth entowerly erigenating from next to nothing and celescalating the himals and all hierarchitectitiptitoploftical with a burning bush abob off its baubletop and with larrons o toolers clittering up and tombles a buckets clottering down of the first was he to bare arms and a name wassaily booslaeugh of riesengeborg his crest of huroldry in vert with ancillars troublant argent a hegoak poursuivant horrid horned his scutschum fessed with archers strung helio of the second hootch is for husbandman handling his hoe hohohoho mister finn youre going to be mister finnagain comeday morm and o youre vine senddays eve and ah youre vinegar hahahaha mister funn youre going to be fined again

Appendix B. Encoded Messages

To make it easier to see the space symbol, the white space character is replaced with '␣' in the following texts. However, the white space remains

unchanged when experiments were performed.

Appendix B.1. f_45.txt

mlsvcqolxvslqmugltvltbuglblqvvulbtlblncwpgrtlsemrelmnlg_tdgjgqalmxt
gdgntmxilwctlbqjvntlgxtmdgqalbrbogjmrlmxlxbtcdgltemnlmnlteglncwpgrtl
vylteglgxgdigtmrnlvylrvjfctmxilsglsbxtltvlboodgnnlteglhcgntmvxlevsljcrelgx
gdialjcntlwglcngolmxlrbddamxilvctlblrvjfctbtmvxltemnlovgnxtlnvcxolbqqlt
ebtlbrbogjmrlbytgdlbqqlblygbtcdglvyljvntljvogjljbremxgnlmnltebtltegmdlg
xgdialrvxncjftmvxlsegxltegaldcxlzgdalybntlmnlhcmtglrvxnmogdbwqglbxolv
xglvylteglqmjmtbtmvxnlvylteglybntgntljbremxgnlmnlteglnfggolbtlsemrelsg
lrbxlodbmxlvyylteglegbtligxgdbtgolmxltegmdlrvjfvxgxtnlncrelbnltdbxnmnt
vdnlocdmxilvfgdbtmvxltegldgbnvxlmlebzglognrdmwgolvcdlncwpgrtlbnlbrbo
gjmrlmnlwgrbcnglsglbdglbrtcbqqalivmxiltvlbnulbxvtegdlvylvcdlycxobjgxtb
qlhcgntmvxnlsebtlmnltegljmxmjcjlgxgdialdghcmdgoltvlrbddalvctlblrvjfctbt
mvxltvlmxtdvocrgltegngljvdglfeanmrbqlbnfgrtnlvylvcdlncwpgrtlmlsmqqldgt
cdxltvlteglymgqolrvzgdgolmxlteglqbntlrebftgdlxbjgqaltegltegvdalvylmxyvdj
btmvxlmtlmnlfvnnmwqgltvltdgbtltemnlncwpgrtlydvjlblntdmrtqalfeanmrbq
lzmgsfvmxtlbxolmtlmnltemnltebtlsmqqljbugltеglqmxulsmtelteglgxgdialvylr
vjfctbtmvxltvlwgimxlsmtelmlsvcqolqmugltvltdaltvlimzglavclbxlcxogdntbxo
mxilvylteglfeanmrbqlogymxmtmvxlvylteglmxyvdjbtmvxlrvxtgxtlvylbljgnnb
igltebtlfeanmrnlnevcqoligtlmxzvqzgolmxltemnlbdgblmnlebdoqalncdfdmnmx
ildgjgjwgdlnebxxvxlsbnlmxmtmbqqalmxtgdgntgolmxlngxomxiljgnnbignlovs
xldgbqlsmdgnlbxolsglrbxxvtlngxoljgnnbignlvylbxalumxolsmtevctlnvjglmxtg
dygdgxrglydvjlteglfeanmrbqlsvdqolmlbjlivmxiltvlmqqcntdbtgltemxinlwalrv
xrgxtdbtmxilvxlblfbdtmrcqbdlzgdalwbnmrlfeanmrbqljvogqlvylbljgnnbiglwg
mxilngxtlmlsbxtlavcltvlzmncbqmkglteglljgnnbiglrvjmxilmxlbnlblnghcgxrglv
ylwv_gnlgbrelvylsemrelrvxtbmxnlblnmxiqglbtvjlmxlgbrelwv_lteglbtvjlrbxlw
glmxlvxglvyltsvlfqbrgnlvxlteglqgytlvdltegldmietlnmoglmylmtnlvxlteglqgytl
tebtlrvcxtnlbnlblkgdvlwmtlmylmtnlvxltegldmietlmtnlblvxglnvlteglntdgbjlv
ylwv_gnlrvjgnlfbntljglbxolwalqvvumxiltvlngglsegdglgbrelbtvjlmnlmlrbxlsvd
ulvctlteglrvddgnfvxomxilwmtltvlngglevsltemnljvogqlrbxlegqflcnlcxogdntbx
olmxyvdjbtmvxlsglebzgltvlqvvulbtlteglfeanmrnlvylpmiiqmxilbtvjnlbdvcxolt
emnldghcmdgnlcnltvlrvxnmogdlteglfeanmrnlvylibngnlnvlmlsmqqlwgimxlwa
ltbumxilblygsltemxinlmlxggolydvjltebtlqgtlcnlwgimxlwalncffvnmxilsglebzgl
blibnlrvxtbmxmxilxlbtvjnlvdljvqgrcqgnlvrrcfamxilblzvqcjglzlvxglsglsmqqltb
ugltemnlibnltvlwglbxlg_rgftmvxbqqalnmjfqglvxglgbrelbtvjlvdljvqgrcqglsmte
mxlmtlsgltbugltegltgdjnltvlwglmxtgdrebxigbwqglegdglmnlgnngxtmbqqalyd
ggltegdglbdglxvlyvdrgnlvylbttdbrtmvxlvdldgfcqnmvxlwgtsggxlgbrelrvxntm

tcgxtltemnlmnlbrtcbqqalblivvolbffdv_mjbtmvxlbtljvogdbtgqalqvslfdgnncdg
nlmlbjlxvslivmxiltvlnedmxulteglibnlfcnemxilbibmxntlmtnlzvqcjglsmtelblfm
ntvxlrvjfdgnnmxilmtltvlzvqcjglzltsvlmlovlbqqltemnlmnvtegdjbqqaltebtlmnl
mlmjjgdngllteglsevqglnantgjlmxlbltegdjbqlwbtelbtlblym_goltgjfgdbtcdgltlnv
ltebtltegltgjfgdbtcdglvyljalbffbdbtcnldgjbmxnlrvxntbxtlmnxtlmtlsvxogdycq
ltebtltemnlebnlbxatemxiltvlovlsmtelsebtlsgdgltbqumxilbwvctlmjlivmxiltvln
evslavclevslymdntlsglsbxtltvluxvslevsljcrelsvdulslmtltbugnltvlrvjfdgnnltegli
bnlxvslblntbxobdoldgncqtlmxljgrebxmrnlebnlmtltebtlmylblyvdrglyljvzgnlte
dvcielblnjbqqlomntbxrglo_lteglsvdulovxgloslmnlyo_lmylteglfdgnncdglvyltegl
ibnlmnlflbxolteglrdvnnlngrtmvxbqlbdgblvylteglfmntvxlmnlblsglrbxldgsdmt
gltemnlcnmxilylghcbqnlfblbxolqgttmxilteglzvqcjglrebxiglvylteglibnlozlghcbq
nlbo_lnvltebtloslmnlfozlxvslsglodbslvxlblntbxobdoldgncqtlydvjlibnltegvdaly
vdlbxlmogbqlibnlbtlfdgnncdglflzvqcjglzlbxoltgjfgdbtcdgltlsglebzgltegldgqbt
mvxlfzlghcbqnlxutlsegdglxlmnlteglxcjwgdlvyljvqgrcqgnlmxlteglibnlbxolulm
nlwvqtkjbxxnlrvxntbxtlbnltlmnlrvxntbxtlvcdlmnvtegdjbqlbnncjftmvxlsglrb
xlfgdyvdjlblnmjfqglmxtgidbtmvxltvlymxolslnmxrglzltsvlmnlnjbqqgdltebxlzl
vxgltemnlhcbxtmtalmnlxgibtmzglbxoltemnlmnlpcntlbldgncqtlvylteglrvxzgx
tmvxltebtlsvdulovxglvxlblibnldbtegdltebxlwalmtlebnlbljmxcnlnmixlxvslvdo
mxbdmqalsegxlsglrvjfdgnnlblibnlsglegbtlmtlcfltemnlmnlbldgncqtlvylmtnlrv
xntmtcgxtlbtvjnlnfggomxilcflbxolibmxmxilumxgtmrlgxgdialevsgzgdlmxlvcd
lrbnglmylsglg_bjmxgltegljvqgrcqgnlvylteglibnlwgyvdglbxolbytgdlrvjfdgnnm
vxlsglymxolxvlomyygdgxrgltegdglbdglteglnbjglxcjwgdlbxoltegalbdglpmiiqm
xilbwvctlxvljvdglvdlqgnnlgxgdigtmrbqqaltebxltegalsgdglwgyvdgltegdglmnlx
vlomyygdgxrglwgtsggxltegltsvlbtltegljvqgrcqbdlqgzgqlnvlsegdglomolteglsvd
ulivlsglfctlnvjglmxltvlrvjfdgnnlteglibnlbxolrvxngdzbtmvxlvylgxgdialnbanlm
tleboltvlivlnvjgsegdglmxlybrtlmtlsbnlrvxzgdtgolmxtvlmxtgdxbqlibnlegbtlw
ctlsbnlfdvjftqalodbmxgolvyylmxtvltegltegdjbqlwbteluggfmxilteglibnlbtltegl
nbjgltgjfgdbtcdgltemnlmnlbrtcbqqalsebtlsgljgbxlwalmnvtegdjbqlrvjfdgnnm
vxlsglovlteglrvjfdgnnmvxlnqvsqalgxncdmxiltebtlbtlbqqltmjgnlteglibnlbxolte
glncddvcxomxilwbtelbdglmxltegdjbqlghcmqmwdmcj

Appendix B.2. h_45.txt

dfw_xkwdflwksflmczwjywxsdvcfwyjsfwnfhbctwkfhbcwgcfcwnfjskwtjwl_z
wtr_twtrczwgcfcwncfycptmzwxjfd_mwtr_xqwzjswbcfzwdsprwtrczwgcfcwtrc
wm_ltwncjnmcwzjskwcancptwtjwvcwhxbjmbckwhxw_xztrhxowltf_xocwjfwd
zltcfhjslwvcp_slcwtrczwisltwkhkxtwrjmkwghtrwlsprwxjxlcxlcwdfwksflmczw
g_lwtrcwkhfcptjfwjyw_wyhfdwp_mmckwofsxxhxolwgrhprwd_kcwkfhmmlwrc
wg_lw_wvhowvccyzwd_xwghtrwr_fkmzw_xzwxcpqw_mtrjsorwrcwkhkwr_bcw

_wbcfzwm_focwdslt_prcwdflwksflmczwg_lwtrhxw_xkwvmjxkcw_xkwr_kwxc_f
mzwtghpcwtrcwsls_mw_djsxtwjywxcpqwgrhprwp_dcwhxwbcfzwslcysmw_lwl
rcwlncxtwljwdsprwjywrcfwthdcwpf_xhxowjbcfwo_fkcxwycxpclwlnzhxowjxw
trcwxchorvjflwtrcwksflmczlwr_kw_wld_mmwljxwp_mmckwkskmczw_xkwhx
wtrchfwjnhxhjxwtrcfcwg_lwxjwyhxcfwvjzw_xzgrcfcwtrcwksflmczlwr_kwcbcf
ztrhxowtrczwg_xtckwvstwtrczw_mljwr_kw_wlcpfctw_xkwtrchfwofc_tcltwyc_f
wg_lwtr_twljdcvjkzwgjsmkwkhlpjbcfwhtwtrczwkhkxtwtrhxqwtrczwpjsmkwv
c_fwhtwhyw_xzjxcwyjsxkwjstw_vjstwtrcwnjttcflwdflwnjttcfwg_lwdflwksflmc
zlwlhltcfwvstwtrczwr_kxtwdctwyjfwlcbcf_mwzc_flwhxwy_ptwdflwksflmczwn
fctcxkckwlrcwkhkxtwr_bcw_wlhltcfwvcp_slcwrcfwlhltcfw_xkwrcfwojjkwyjfw
xjtrhxowrslv_xkwgcfcw_lwsxksflmczhlrw_lwhtwg_lwnjllhvmcwtjwvcwtrcwks
flmczlwlrskkcfckwtjwtrhxqwgr_twtrcwxchorvjflwgjsmkwl_zwhywtrcwnjttcfl
w_ffhbckwhxwtrcwltfcctwtrcwksflmczlwqxcgwtr_twtrcwnjttcflwr_kw_wld_m
mwljxwtjjwvstwtrczwr_kwxcbcfwcbcxwlccxwrhdwtrhlwvjzwg_lw_xjtrcfwojjk
wfc_ljxwyjfwqccnhxowtrcwnjttcflw_g_zwtrczwkhkxtwg_xtwkskmczwdhahxo
wghtrw_wprhmkwmhqcwtr_twgrcxwdfw_xkwdflwksflmczwgjqcwsnwjxwtrcw
ksmmwof_zwtsclk_zwjsfwltjfzwlt_ftlwtrcfcwg_lwxjtrhxow_vjstwtrcwpmjskzw
lqzwjstlhkcwtjwlsoocltwtr_twltf_xocw_xkwdzltcfhjslwtrhxolwgjsmkwljjxwvc
wr_nncxhxow_mmwjbcfwtrcwpjsxtfzwdfwksflmczwrsddckw_lwrcwnhpqckwjs
twrhlwdjltwvjfhxowthcwyjfwgjfqw_xkwdflwksflmczwojllhnckw_g_zwr_nnhm
zw_lwlrcwgfcltmckw_wlpfc_dhxowkskmczwhxtjwrhlwrhorwpr_hfwxjxcwjywt
rcdwxjthpckw_wm_focwt_gxzwjgmwymsttcfwn_ltwtrcwghxkjgw_twr_mywn_l
twchortwdfwksflmczwnhpqckwsnwrhlwvfhcyp_lcwncpqckwdflwksflmczwjxw
trcwprccqw_xkwtfhckwtjwqhllwkskmczwojjkwvzcwvstwdhllckwvcp_slcwksk
mczwg_lwxjgwr_bhxow_wt_xtfsdw_xkwtrfjghxowrhlwpcfc_mw_twtrcwg_mml
wmhttmcwtzqcwprjftmckwdfwksflmczw_lwrcwmcytwtrcwrjslcwrcwojtwhxtj
wrhlwp_fw_xkwv_pqckwjstwjywxsdvcfwyjsflwkfhbcwhtwg_lwjxwtrcwpjfxcfw
jywtrcwltfcctwtr_twrcwxjthpckwtrcwyhfltwlhoxwjywljdctrhxowncpsmh_fw_
wp_twfc_khxow_wd_nwyjfw_wlcpjxkwdfwksflmczwkhkxtwfc_mhecwgr_twrcw
r_kwlccxwtrcxwrcwicfqckwrhlwrc_kw_fjsxkwtjwmjjqw_o_hxwtrcfcwg_lw_wt_
vvzwp_twlt_xkhxowjxwtrcwpjfxcfwjywnfhbctwkfhbcwvstwtrcfcwg_lxtw_wd_
nwhxwlhortwgr_twpjsmkwrcwr_bcwvccxwtrhxqhxowjywhtwdsltwr_bcwvccx
w_wtfhpqwjywtrcwmhortwdfwksflmczwvmhxqckw_xkwlt_fckw_twtrcwp_twh
twlt_fckwv_pqw_lwdfwksflmczwkfjbcw_fjsxkwtrcwpjfxcfw_xkwsnwtrcwfj_kw
rcwg_tprckwtrcwp_twhxwrhlwdhffjfwhtwg_lwxjgwfc_khxowtrcwlhoxwtr_twl
_hkwnfhbctwkfhbcwxjwmjjqhxow_twtrcwlhoxwp_tlwpjsmkxtwfc_kwd_nlwjf
wlhoxlwdfwksflmczwo_bcwrhdlcmyw_wmhttmcwlr_qcw_xkwnstwtrcwp_twjs
twjywrhlwdhxkw_lwrcwkfjbcwtjg_fkwtjgxwrcwtrjsortwjywxjtrhxowcapcntw

_wm_focwjfkcfwjywkfhmmlwrcwg_lwrjnhxowtjwoctwtr_twk_zwvstwjxwtrcw
ckocwjywtjgxwkfhmmlwgcfcwkfhbcxwjstwjywrhlwdhxkwvzwljdctrhxowcml
cw_lwrcwl_twhxwtrcwsls_mwdjfxhxowtf_yyhpwi_dwrcwpjsmkxtwrcmnwxjth
phxowtr_twtrcfcwlccdckwtjwvcw_wmjtwjywltf_xocmzwkfcllckwncjnmcw_vjs
twncjnmcwhxwpmj_qlwdfwksflmczwpjsmkxtwvc_fwncjnmcwgrjwkfcllckwhx
wysxxzwpmjtrclwtrcwoctsnlwzjswl_gwjxwzjsxowncjnmcwrcwlsnnjlckwtrhlw
g_lwljdcwltsnhkwxcgwy_lrhjxwrcwkfsddckwrhlwyhxocflwjxwtrcwltccfhxow
grccmw_xkwrhlwczclwycmmwjxw_wrskkmcwjywtrclcwgchfkjlwlt_xkhxowus
htcwpmjlcwvzwtrczwgcfcwgrhlncfhxowcaphtckmzwtjoctrcfwdfwksflmczwg_l
wcxf_ockwtjwlccwtr_tw_wpjsnmcwjywtrcdwgcfcxtwzjsxow_tw_mmwgrzwtr_t
wd_xwr_kwtjwvcwjmkcfwtr_xwrcwg_lw_xkwgc_fhxow_xwcdcf_mkwofccxwpm
j_qwtrcwxcfbcwjywrhdwvstwtrcxwhtwltfspqwdfwksflmczwtr_twtrhlwg_lwnf
jv_vmzwljdcwlhmmzwltsxtwtrclcwncjnmcwgcfcwjvbhjslmzwpjmmcpthxowy
jfwljdctrhxowwzclwtr_twgjsmkwvcwhtwtrcwtf_yyhpwdjbckwjxw_xkw_wycg
wdhxstclwm_tcfwdfwksflmczw_ffhbckwhxwtrcwofsxxhxolwn_fqhxowmjtwrh
lwdhxkwv_pqwjxwkfhmmlwdfwksflmczw_mg_zlwl_twghtrwrhlwv_pqwtjwtrc
wghxkjgwhxwrhlwjyyhpcwjxwtrcwxhxtrwymjjfwhywrcwr_kxtwrcwdhortwr_
bcwyjsxkwhtwr_fkcfwtjwpjxpcxtf_tcwjxwkfhmmlwtr_twdjfxhxowrcwkhkxtw
lccwtrcwjgmlwlgjjnhxown_ltwhxwvfj_kwk_zmhortwtrjsorwncjnmcwkjgxwhx
wtrcwltfcctwkhkwtrczwnjhxtckw_xkwo_eckwjncxwdjstrckw_lwjgmw_ytcfwjg
mwlnckwjbcfrc_kwdjltwjywtrcdwr_kwxcbcfwlccxw_xwjgmwcbcxw_twxhortt
hdcwdfwksflmczwrjgcbcfwr_kw_wncfycptmzwxjfd_mwjgmwyfccwdjfxhxowrc
wzcmmckw_twyhbcwkhyycfcxtwncjnmcwrcwd_kcwlcbcf_mwhdnjft_xtwtcmc
nrjxc

Appendix B.3. j_45.txt

fqksff_nupxzousksuxnluxlxtzuvfgtuzrsfksugvuzjgfsuogucsnlugvucxducfqn
izu_zucduxubgttglq_zukqb_zugvufsbqfb_exoqgnucxbwuogujgrojubxzoesuxnl
usnkqfgnzuzqfuofqzofxtukqgesfulxtgfszuvfgksfuojsuzjgfouzsxujxlupxzzsnbgf
sufsxffqksluvfgtungfojuxftgfqbxugnuojqzuzqlsuojsuzbfxiiduqzojt_zugvus_fgp
sutqngfuogurqselsfvqijoujqzupsnqzgexosurxfungfujxluogpzxrdsfzufgbwzucdu
ojsuzofsxtugbgnssusaxiisfxosluojstzsezsuoguex_fsnzubg_nodzuigfiqgzurjqesu
ojsdursnoulg_ceqnuojsqfut_tpsfuxeeuojsuoqtsungfuxkgqbsuvfgtuxvqfsucseeg
rzslutqzjsutqzjsuoguox_vox_vuoj_xfopsxofqbwungoudsouojg_ijuksnqzzggnux
vosfujxluxuwqlzbxluc_oosnlsluxucexnlugeluqzxxbungoudsouojg_ijuxeezuvxq
fuqnukxnszzdursfsuzgzqsuzszojsfzurfgojurqojuorgnsunxojxnlmgsufgouxupsb
wugvupxzutxeoujxlumjstugfuzjsnucfsrslucduxfbeqijouxnlufgfdusnluoguojsuf
siiqncfgrurxzuogucsuzssnufqnizgtsugnuojsuxh_xvxbsuojsuvxeeucxcxcxlxeijx

fxijoxwxttqnxffgnnwgnncfgnnognnsfufgnno_gnnoj_nnofgkxffjg_nxrnzwxrnog
gjggjggflsnsnoj_fn_wugvuxugnbsurxeezofxqougelpxffuqzufsoxeslusxfeduqnuc
sluxnluexosfugnueqvsulgrnuojfg_ijuxeeubjfqzoqxnutqnzofsezduojsuifsxouvxe
eugvuojsugvvrxeeusnoxqesluxouz_bjuzjgfoungoqbsuojsupvomzbj_osugvuvqn
nsixnusfzsuzgeqlutxnuojxouojsuj_tpodjqeejsxlugvuj_tzsevupf_tpoeduzsnlzux
nu_nh_qfqniugnsurseeuoguojsurszouqnuh_szougvujqzuo_tpodo_togszuxnluojs
qfu_po_fnpqwspgqnoxnlpexbsuqzuxouojsuwngbwug_ouqnuojsupxfwurjsfsugf
xniszujxksucssnuexqluoguf_zou_pgnuojsuifssnuzqnbsulskeqnzvqfzouegkslueq
kkdurjxoubexzjszujsfsugvurqeezuisnurgnozugdzofdiglzuixiiqnuvqzjduiglzucf
swwswuwswwswuwswwswuwswwswuwgxauwgxauwgxau_xe_u_xe_u_xe_uh_x
g_x_jurjsfsuojsucxllsexfqszupxfoqzxnzuxfsuzoqeeug_ouogutxojtxzosfutxexbj_
zutqbifxnszuxnluojsuksflgnzubxoxpseoqniuojsubxtqcxeqzoqbzug_ougvuojsur
jgdoscgdbsugvujgglqsujsxluxzzqsixoszuxnlucggtsfqnizofgtzuzgluzucfgglucsu
tsuvsxfuzxniegfqxnzuzxksuxftzuxpsxeurqojuexftzuxppxeeqniuwqeedwqeewq
eeduxuogeeuxuogeeurjxoubjxnbsub_llesdzurjxoubxzjsezuxqfsluxnluksnoqexo
slurjxoucqlqtsogegkszuzqnl_bslucdurjxouosigosoxczgeksfzurjxouof_suvsseqni
uvgfuojsqfuzujxdxqfurqojurjxouzofxrniukgqbsugvuvxezsumqbb_pugujsfsujsf
sujgrujgojuzpfgreslutsouojsul_zwouojsuvxojsfugvuvgfnqbxoqgnqzozuc_ougut
duzjqnqniuzoxfzuxnlucgldujgrujxojuvxnszpxnnslutgzoujqijujsxksnuojsuzwd
zqinugvuzgvouxlksfoqzstsnouc_ourxzuqyuqzs_ousfsursfsuzsrsfzuojsugxwzugv
uxelungruojsdueqsuqnupsxoudsousetzuesxpurjsfsuxzwszuexdupjxeeuqvudg_
uc_ourqeeufqzsudg_ut_zouxnlungnsuzguzggnusqojsfuzjxeeuojsupjxfbsuvgfuo
jsun_nbsubgtsuoguxuzsolgrnuzsb_exfupjgsnqzjucditszosfuvqnnsixnugvuojsu
zo_oosfqniujxnluvfsstsnuzutx_fsfueqksluqnuojsucfgxlszourxduqttxfiqnxcesuq
nujqzuf_zjeqouoggvxfcxbwuvgfutszz_xiszucsvgfsumgzj_xnum_liszujxluiqksnu
_zun_tcsfzugfujsekqoqb_zubgttqoosluls_osfgngtdugnsudsxzodlxdujsuzosfnsed
uzof_awujqzuososuqnuxuo_cuvgfuogurxozbjuojsuv_o_fsugvujqzuvxoszuc_ous
fsujsuzrqvoeduzoggwuqoug_ouxixqnucduojsutqijougvutgzszuojsuksfdurxosfu
rxzuskqpxfxosluxnluxeeuojsui_snnszszujxlutsouojsqfusagl_zuzguojxoug_ijouo
guzjgrudg_urjxouxupsnozbjxnms_bjdubjxpujsurxzuxnlul_fqniutqijoduglludsx
fzuojqzutxnugvujglubstsnouxnluslqvqbszuqnuogpsfuzuojgfpupqesluc_qel_niu
z_pfxuc_qel_niupgnuojsucxnwzuvgfuojsueqksfzucduojsuzgxnizgujsuxllesueql
lesupjqvqsuxnnqsu_iisluojsueqooesubfxdoj_furqojsfujxdfsuqnujgnlzuo_bwu_p
udg_fupxfouqnjsfugvorjqesucxec_eg_zutqojfsuxjsxlurqojuiggleduofgrseuqnuif
xzpuxnluqkgfgqeslugksfxeezurjqbjujsujxcqoxb_exfeduvgnlzsslueqwsujxfg_nu
bjqelsfqbusiiscsfojujsurg_elubxeqi_exosucdut_eoqpeqbxceszuojsuxeeoqo_lsux
nlutxeeoqo_lsu_noqeujsuzsszxrucdunsxoeqijougvuojsueqh_gfurjsfsorqnuuorxz
ucgfnujqzufg_nljsxluzoxpesugvugojsfulxdzuogufqzsuqnu_nlfszzutxqzgnfdu_p

zoxnlslumgdifxnoqouxurxxergfojugvuxuzwdsfzbxpsugvutgzousdsv_eujgdojus
nogrsfedusfqisnxoqniuvfgtunsaouogungojqniuxnlubseszbxexoqniuojsujqtxez
uxnluxeeujqsfxfbjqosboqoqpoqogpegvoqbxeurqojuxuc_fnqniuc_zjuxcgcugvvu
qozucx_cesogpuxnlurqojuexffgnzuguoggesfzubeqoosfqniu_puxnluogtceszuxuc
_bwsozubegoosfqniulgrnugvuojsuvqfzourxzujsuogucxfsuxftzuxnluxunxtsurxz
zxqeducggzexs_ijugvufqszsniscgfiujqzubfszougvuj_fgelfduqnuksfourqojuxnbq
eexfzuofg_cexnouxfisnouxujsigxwupg_fz_qkxnoujgffqlujgfnslujqzuzb_ozbj_tuv
szzslurqojuxfbjsfzuzof_niujseqgugvuojsuzsbgnlujggobjuqzuvgfuj_zcxnltxnujx
nleqniujqzujgsujgjgjgjgutqzosfuvqnnudg_fsuigqniuogucsutqzosfuvqnnxixqnu
bgtslxdutgftuxnlugudg_fsukqnsuzsnllxdzusksuxnluxjudg_fsukqnsixfujxjxjxjx
utqzosfuv_nnudg_fsuigqniuogucsuvqnsluxixqn

Appendix C. Python Code

Appendix C.1. Data Mining

```
"""
do data mining and obtain the one-point and two-point
                                statistics about the English
                                language, and then calculate
                                the likelihood function
"""
import re
import numpy as np
import matplotlib.pyplot as plt


legal_chars = 'abcdefghijklmnopqrstuvwxyz '
chars_to_index = {c:i for i, c in enumerate(legal_chars)}
index_to_chars = {i:c for i, c in enumerate(legal_chars)}


def load_data(file='pride-and-prejudice.txt'):
    """
    load the data from the file remove all the punctuations
                                    and make all the words
                                    lower case, only leave
                                    characters a-z and space

    return: one long string
    """
    with open(file, 'r') as f:
        data = f.read()
```

```
    # to lowercase
    data = data.lower()
    # filter out white spaces
    data = data.replace('\n', ' ')
    data = data.replace('\r', ' ')
    data = data.replace('\t', ' ')
    # only keep the legal characters
    data = ''.join([c for c in data if c in legal_chars])
    # merge white spaces
    data = re.sub(' +', ' ', data)

    return data


def get_one_point_statistics(data):
    """
    get the one-point statistics about the English language
                                    from input data
    args:
        data: a long string
    return:
        a dictionary with keys as characters and values as
                                        the number of times
    """
    # get the frequency of each character
    freq = {}
    for c in legal_chars:
        freq[c] = data.count(c)
    # get the total number of characters
    total = sum(freq.values())
    # get the probability of each character
    for c in freq:
        freq[c] /= total

    # make sure this is a probability distribution
    assert abs(sum(freq.values()) - 1.0) < 1e-6

    return freq


def get_transition_matrix(data):
    """
    get the transition matrix from input data
    args:
        data: a long string
```

```
    return:
        a matrix Q indexing the transition probabilities
        Q(x, y) is the probability of transitioning from x to
                                            y
    """
    Q = np.zeros((len(legal_chars), len(legal_chars)))
    # count tuples
    for i in range(len(data) - 1):
        x = chars_to_index[data[i]]
        y = chars_to_index[data[i+1]]
        Q[x, y] += 1
    # normalize
    Q = Q / Q.sum(axis=1, keepdims=True)
    # make sure each row is a probability distribution
    assert np.isclose(Q.sum(axis=1), 1.0).all()

    return Q


def visualize(freq, Q):
    """
    visualize the one-point frequencies and the transition
                                        matrix Q
    """
    plt.imshow(np.array(list(freq.values()), dtype=np.float32
                                        ).reshape((1, -1)), cmap='
                                        Blues')
    plt.colorbar(orientation="horizontal")
    plt.xticks(range(len(freq)), list(freq.keys()))
    plt.yticks([], [])
    plt.show()

    plt.imshow(Q, cmap='Blues')
    plt.colorbar()
    plt.xticks(range(len(freq)), list(freq.keys()))
    plt.yticks(range(len(freq)), list(freq.keys()))
    plt.show()
    plt.close()


if __name__ == '__main__':
    # for debugging and plotting purposes
    data = load_data()
    print(len(data))
```

```
    freq = get_one_point_statistics(data)
    print(freq)
    print(len(freq))

    Q = get_transition_matrix(data)
    print(Q)
    print(Q.shape)

    visualize(freq, Q)
```

Appendix C.2. Metropolis Algorithm and Decoding

```
"""
this is where the magic happens. This file will decode the
                                message encoded by the
                                substitution cipher
"""
import argparse
import random
import math
from copy import deepcopy

import matplotlib.pyplot as plt
import numpy as np

from data_mining import get_one_point_statistics,
                                get_transition_matrix, \
    load_data, legal_chars, chars_to_index, index_to_chars


def load_encoded_message(file):
    """
    load the encoded message from a file into a long string
    """
    with open(file, 'r') as f:
        data = f.read()
    return data


def swap_element(permutation):
    """
    swap the elements at index i and j in permutation
    this swapped permutation is defined as a neighbor of the
                                    current
    permutation in the MCMC random graph
```

```
    """
    # choose two indices to swap
    i, j = np.random.choice(len(permutation), 2, replace=
                                    False)

    neighbor = list(deepcopy(permutation))
    neighbor[i], neighbor[j] = neighbor[j], neighbor[i]
    return ''.join(neighbor)


def energy_func(encoded_msg, one_point_stat, transition_mat,
                                permutation):
    """
    get the energy of a certain permutation
    the energy is defined as the negative log likelihood
    likelihood is P(permutation | encoded_msg)
    """
    permutation_inverse = lambda b: legal_chars[permutation.
                                    index(b)]
    likelihood  = 0
    # liklihood of the first letter
    likelihood += math.log(one_point_stat[permutation_inverse
                                    (encoded_msg[0])])
    # likelihood of the rest of the transition pairs
    for i in range(1, len(encoded_msg)-1):
        x = permutation_inverse(encoded_msg[i]),
        y = permutation_inverse(encoded_msg[i+1])
        x_idx, y_idx = chars_to_index[x], chars_to_index[y]
        try:
            likelihood += math.log(transition_mat[x_idx,
                                            y_idx])
        except ValueError:
            # transition probability is 0
            likelihood += math.log(1e-16)
    return -likelihood


def mcmc(num_iters, beta, encoded_msg, one_point_stat,
         transition_mat,
         plot_save_path,
         start_state='abcdefghijklmnopqrstuvwxyz ',
         plot_every=1):
    """
    markov chain monte carlo
```

```
try to sample the correct permutation according to a
                                Gibbs distribution
make a plot of the energy of walk throught the random
                                graph, record the energy
every 'plot_every' iterations
"""
current_state = start_state
energy = lambda state: energy_func(encoded_msg,
                                one_point_stat,
                                transition_mat, state)
energy_list = []
# start iteration
for i in range(num_iters):
    print(f"iteration {i}")

    # record the energy
    if i % plot_every == 0:
        energy_list.append(energy(current_state))

    # choose a neighbor of current state uniformly at
                                    random
    next_state = swap_element(current_state)

    # accept or reject according to energy change
    energy_diff = energy(next_state) - energy(
                                    current_state)
    if energy_diff < 0:
        current_state = next_state
    else:
        accept_prob = math.exp(-beta * energy_diff)
        if random.random() < accept_prob:
            # accept
            current_state = next_state
        else:
            # reject
            pass

# plot the energy trajectory
plt.plot(plot_every * np.arange(len(energy_list)),
                                energy_list)
plt.xlabel('Steps')
plt.ylabel('Energy')
plt.title('Energy of Walk Through Metropolis Graph')
plt.savefig(plot_save_path)
```

```
    return current_state


def decode_message(encoded_msg, permutation, save_to=None):
    """
    decode a long string
    args:
        encoded_msg: a long string encoded by substitution
                                        cipher
        permutation: a 27-char string of all legal chars
    return:
        decoded_msg: a long string decoded by substitution
                                        cipher
    """
    permutation_inverse = lambda b: legal_chars[permutation.
                                    index(b)]
    decoded_msg = ''.join(map(permutation_inverse,
                                    encoded_msg))

    # save the result
    if save_to:
        with open(save_to, 'w') as f:
            f.write(decoded_msg)

    return decoded_msg


def main():
    """
    put everything together
    """
    parser = argparse.ArgumentParser()
    parser.add_argument('--data', type=str, default='pride-
                                    and-prejudice.txt', help='
                                    the file containing the
                                    data of true English
                                    language')
    parser.add_argument('--code', type=str, default='f_45.txt
                                    ', help='the file
                                    containing the encoded
                                    message')
    parser.add_argument('--save_path', type=str, default='
                                    decoded.txt', help='the
                                    file to save the decoded
                                    message')
```

```
    parser.add_argument('--beta', type=float, default=1.0,
                                    help='the beta parameter
                                    of the Gibbs distribution'
                                    )
    parser.add_argument('--num_iters', type=int, default=
                                    100000, help='the number
                                    of iterations to run MCMC'
                                    )
    args = parser.parse_args()

    # set random seed
    random.seed(0)
    np.random.seed(0)

    # load the code
    encoded = load_encoded_message(args.code)

    # data mining on the true English language
    true_language_data = load_data(args.data)
    one_point_stat = get_one_point_statistics(
                                    true_language_data)
    transition_matrix = get_transition_matrix(
                                    true_language_data)

    # run MCMC
    final_permutation = mcmc(
        num_iters=args.num_iters,
        beta=args.beta,
        encoded_msg=encoded,
        one_point_stat=one_point_stat,
        transition_mat=transition_matrix,
        plot_save_path=args.save_path[:-4] + '_energy.png',
    )
    print(final_permutation)

    # decode the message
    real_msg = decode_message(encoded, final_permutation,
                                    save_to=args.save_path)
    print(real_msg)


if __name__ == '__main__':
    main()
```